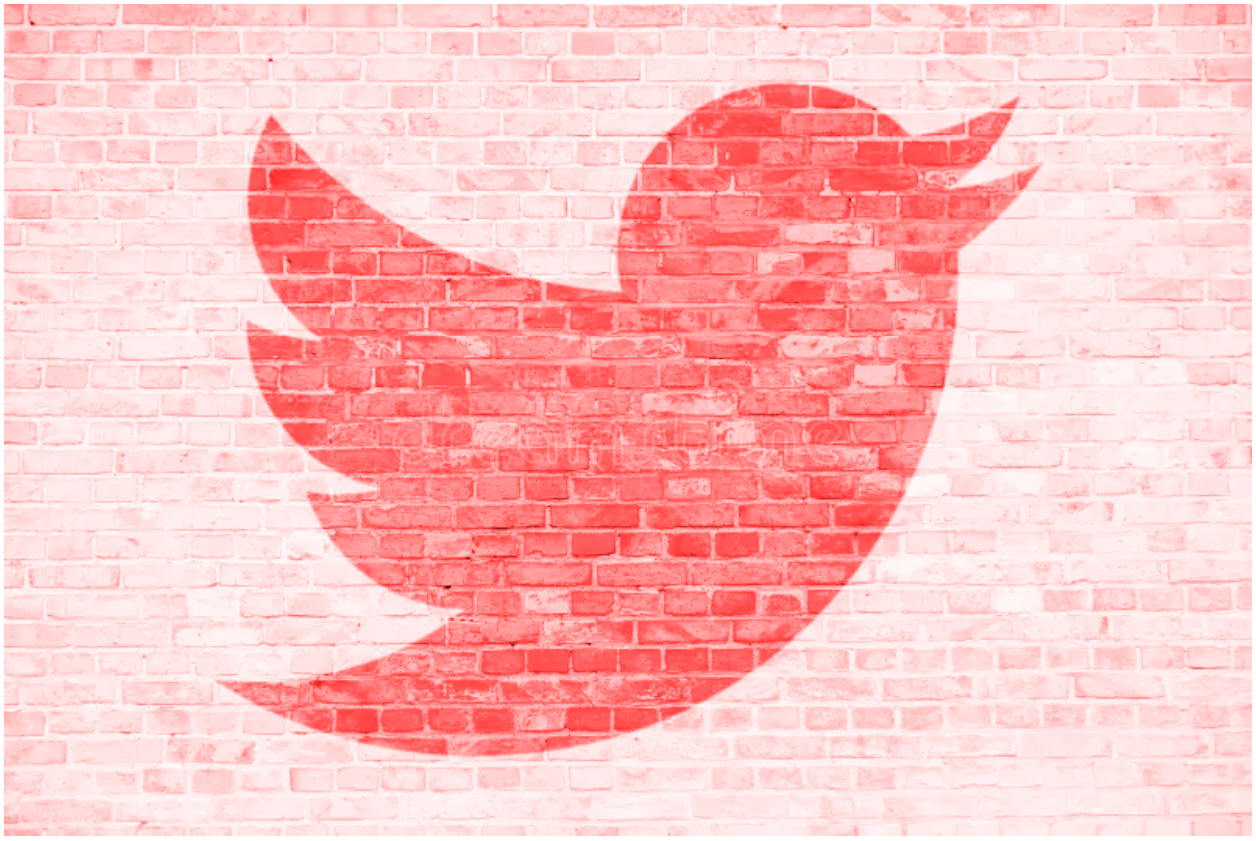

Lab 02

Server-side Development using Python and SQL

TDDD97/732A56

Web Programming

Department of Computer Science

Linköping University

Introduction

In this second lab, you will use Python and SQL to implement the server-side of the final web application in the form of a Rest API. You will use the Flask microframework and the SQLite to handle web requests and data storage respectively.

You are required to develop step by step and implement each step according to the instructions. Once you are finished with each lab, you will present your work to your responsible lab assistant. For more specific information about the presentation and evaluation process of lab 2, please see the section: *Presentation and Evaluation*. For more general information about the examination process, please check the course page.

Requirements

By the end of lab 2, the following requirements shall be met:

Functional:

- All server functions specified in the lab instructions shall work exactly as specified.

Non-Functional:

- The server shall use an SQLite database to store all user data.
- The server shall use appropriate HTTP methods for all routes.
- All the server-side functions shall return their responses in JSON.

The Project Folder

You need to create a new project folder with an arbitrary name. Your project folder shall at least contain the following files by the end of lab 2:

- server.py
- database_helper.py
- database.db
- schema.sql

server.py

This file shall contain all the server side services, implemented using Python and Flask.

database_helper.py

This file will contain all the functions that access and control the database and shall contain some SQL scripts. This file will be used by the *server.py* to access the database. This file shall NOT contain any domain functions like sign in or sign up and shall only contain data-centric functionality like find_user(), remove_user(), create_post() and etc. Implementing sign_in() in server.py shall involve a call to find_user() implemented in database_helper.py .

database.db

This is a SQLite file which will contain your database. Your database is composed of different tables which in turn contain the actual data, such as users' personal information.

schema.sql

This file shall contain the SQL script used to initialize the database. *database_helper.py* or *SQLite3* command tool will use this file to create all the tables and insert the default data. This file should be completed and executed before implementing and running any of the server side procedures.

Development tools

Like in lab 1 you are free to use any text/code editor of your choice. We recommend however VSCode as it has worked smoothly in the course without any problems.You will also need the following tools to complete lab 2.

Python

In order to be able to install the packages you need to complete this lab you will need to create a virtual environment for `Python`, in which you can install additional modules. In this lab you are required to use `Python 3`. You can verify which version of Python you are running by executing the following command in a terminal:

```
python3 -V
```

Now you need to proceed to create a new directory for the virtual environment. `Python` virtual environments are created using the command-line tool `virtualenv`. You can learn how to use `virtualenv` in the documentation found at their website: https://virtualenv.pypa.io/en/latest/user_guide.html. Please note that you shall need `virtualenv` command to install Python 3 as following:

```
virtualenv -p python3 specified_directory
```

Use `virtualenv` to create a new virtual environment in a new directory. `virtualenv` will install a new version of `Python` found in "`<specified_directory>/bin`". `Virtualenv` will also install `pip3`, a tool for managing and installing new packages. The documentation for `pip3` can be found at https://pip.pypa.io/en/stable/. You'll need to use `pip3` in order to install Flask.

NOTE

By using `virtualenv`, we can make a virtual machine with its own settings and libraries which is customized for your web application. This is a common approach to create and use a separate and isolated virtual machine for a new web application.

NOTE

From this point you will work with the executables available in the virtual environment. In order to avoid the need to change directory to where the virtual environment is installed or entering a complete path in order to run the `python` command inside the virtual environment, run first the following command every time you start a new terminal:

`source path_to_virtual_environment/bin/activate`

Now you can only use python3 and pip3 commands in the terminal without any need to provide the full path.

Python documentation:
https://docs.python.org/3/

Flask

Flask is a lightweight web framework written in Python. One of the good features of the Flask framework is that it provides a built-in development web server and debugger. By using Flask you can write your back-end code in Python and run it using the built-in web server. Flask can be added as a module to your already created Python virtual environment using the `pip3` command. The following links provide more information about the Flask framework and how to install it.

Flask official website:
http://flask.pocoo.org/

Information about installing Flask:
http://flask.pocoo.org/docs/installation/

Once you are done you can use the command "`pip3 freeze`" to see which packages have been installed by using `pip3` and verify that Flask has been installed successfully. Make sure that you always run your application using the executables in the virtual environment. If you use the global Python executable, any packages installed by using `pip3` belonging to the virtual environment will not be available.

SQLite

SQLite is a relational database management system which is now widely used for client-side and server-side purposes. It does not provide a separate process and lets you store your data directly in an individual file on disk using SQL language. The version 3 of SQLite is available on the lab systems and accessible via the command `sqlite3`. You can get information about how to use SQLite3 by executing the following command on the lab systems:

```
man sqlite3
```

SQLite official website
http://www.sqlite.org/

Information about SQL language
http://www.w3schools.com/sql/default.asp

Information about using SQLite3 in python programs
http://docs.python.org/2/library/sqlite3.html

Information about using SQLite3 with Flask framework
http://flask.pocoo.org/docs/patterns/sqlite3/

Telnet client

Telnet is a network protocol for bidirectional text exchange over the network. By using telnet client you can get connected to a web server and send http requests to it. Once your request is processed by the server, it returns a text-based http response to the telnet client which is displayed right after. The Telnet client is installed on our lab systems and you can access it by using the telnet command. you can get information about how to use the Telnet client by executing the following command on the lab systems:

```
man telnet
```

Postman

You can also, instead of using telnet, use Postman which can be used for sending HTTP requests using a provided Graphical User Interface. You can download Postman from the following page:

https://www.getpostman.com

HTTP basics

In this lab, you are about to write the server-side functionalities for your Twidder application. This means you need to understand the basics of HTTP, the protocol used to transfer data on the Web. There is a huge amount of resources available online, we suggest taking a look at the following page:

https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP

Make sure you read the parts about **GET, POST, PUT and DELETE** methods to be able to use each of them in its correct place. It is important to have a good understanding about what an HTTP header is and what it contains.

Lab Instructions

The lab has been divided into the following two steps. After completing each server function, you are required to test it using Telnet or Postman.

Step 0: Hello Flask!

In this first step you are required to install the Flask framework on your system according to the instructions available in section 2. Once it's done, you will write a "Hello World" example and test it by using Telnet or any other tools. An example can be found in the documentation provided for Flask.

Step 1: Implementing the Twidder back-end using Python and SQL

In this step you shall implement all functions needed by your front-end application from lab 1. The server functions you create here will often have a direct counterpart in "`serverstub.js`", provided in the previous lab. You will need to create a route for each function, so that it is accessible by the client. Each function, when invoked through its route, is required to return the data in JSON format containing the following information, fields:

1. **success:** true or false, indicating if the function has been executed successfully or not.
2. **message:** A text message describing the success or failure.
3. **data:** The actual returned data.

Example: {"success": true, "message": "Successfully signed in.", "data": token}

NOTE
If needed, the service shall validate the received token to make sure the request is authentic.

To send the required data to the implemented services, you shall follow the following general rules:

1. Use the URL when the used method is GET.
2. Use JSON when the used method is POST, PUT or DELETE.
3. Always use HTTP headers for sending the *token*.

***Authorization* header shall be used for sending and receiving the token.**

The list is formatted as following:
`Function_name`
Description: a description of what the function is intended to do.
Input: a description of what input the server is expecting.
Returned data: either a description of what data the server shall return, or a "-" if no special data is to be returned.
Scenarios: problematic cases which need to be handled.

The list of the functions which shall be implemented at the server-side

`sign_in`

Description: authenticates the username by the provided password.

Input:

- *username*: email address
- *password*

Returned data: a randomly generated access token if the authentication is successful.

Scenarios: [incorrect username or password]

`sign_up`

Description: registers a new user in the database.

Input: seven string values representing the following with exact name of parameters:

- *email*
- *password*
- *firstname*
- *familyname*
- *gender*
- *city*
- *country*

Returned data: -

Scenarios: [invalid data field, user exists]

Note

- This service needs to validate all received parameters to make sure that none of them is empty.
- The password is at least X characters long. X shall be the same as the value used at the client-side.

`sign_out`

Description: signs out a user from the system.

Input:

- *token*: a string containing the access token of the user requesting to sign out.

Returned data: -

Scenarios: [incorrect token]

`change_password`

Description: changes the password of the current user to a new one.
Input:

- *token*: a string containing the access token of the current user.
- *oldpassword*: the old password of the current user.
- *newpassword*: the new password.

Returned data: -
Scenarios: [incorrect token, incorrect oldpassword, invalid newpassword]

`get_user_data_by_token`

Description: retrieves the stored data for the user whom the passed token is issued for. The currently signed in user can use this method to retrieve all its own information from the server.
Input:

- *token*: a string containing the access token of the current user.

Returned data:

- *email*
- *firstname*
- *familyname*
- *gender*
- *city*
- *country*

Scenarios: [incorrect token]

`get_user_data_by_email`

Description: retrieves the stored data for the user specified by the passed email address.
Input:

- *token*: a string containing the access token of the current user.
- *email*: the email address of the user to retrieve data for.

Returned data:

- *email*
- *firstname*
- *family name*
- *gender*
- *city*
- *country*

Scenarios: [incorrect token, email not found]

`get_user_messages_by_token`

Description: retrieves the stored messages for the user whom the passed token is issued for. The currently signed in user can use this method to retrieve all its own messages from the server.
Input:

- *token*: a string containing the access token of the current user.

Returned data: all of the messages sent to the user as an array.
Scenarios: [incorrect token]

`get_user_messages_by_email`

Description: retrieves the stored messages for the user specified by the passed email address.
Input:

- *token*: a string containing the access token of the current user.
- *email*: the email address of the user to retrieve messages for.

Returned data: all of the messages sent to the user as an array.
Scenarios: [incorrect token, email not found]

`post_message`

Description: tries to post a message to the wall of the user specified by the email address.
Input:

- *token*: a string containing the access token of the current user.
- *message*: The message to post.
- *email*: The email address of the recipient.

Returned data: -
Scenarios: [incorrect token, email not found, empty message]

Note
It is your task to decide which HTTP method shall be used for each server function.

One of the important tasks of the backend is to store information more or less permanently. In this step you need to use SQLite3/SQL, using ORMs are also allowed, and python data structures to store certain information at server side, for example information about signed up or signed in users. It's up to you to decide how to store each type of data but you need to defend your choice during presentation. If you are in doubt, discuss your choice of implementation with your lab assistant.

As you proceed, you need to call the implemented procedures using the Telnet client, Postman, your browser or any other tool. Postman gives you the possibility to call server-side procedures with arbitrary arguments and methods and investigate the output.

Questions for consideration

1. What security risks can storing passwords in plain text cause? How can this problem be addressed programmatically?

2. As http requests and responses are text-based information, they can be easily intercepted and read by a third-party on the Internet. Please explain how this problem has been solved in real-world scenarios.

3. How can we use Flask for implementing multi-page web applications? Please explain how Flask templates can help us on the way?

4. Please describe a Database Management System. How SQLite is different from other DBMSs?

5. Do you think the Telnet client is a good tool for testing server-side procedures? What are its possible shortages?

Presentation and Evaluation

Once you are finished with lab 2, you will present your work to your direct lab assistant during a scheduled lab session. You may be asked about the details of your implementation individually.